

Virtual Environments

In Python

What problem are we trying to solve?

If you are the only developer on a project, and you never have to update your code, and you never need to change a version dependency, and you never collaborate with anyone, and you will never run any code on your machine except the code that you have written, then there is no problem to solve.

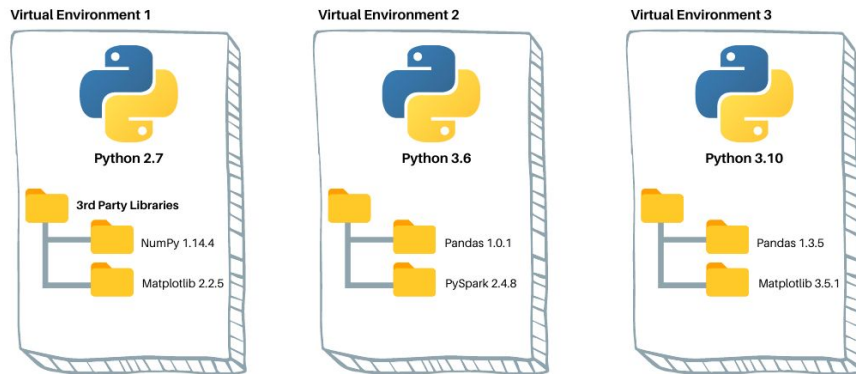
Is this a realistic scenario for a data engineer?

```
npm ERR! code ERESOLVE
npm ERR! ERESOLVE unable to resolve dependency tree
npm ERR!
npm ERR! While resolving: gf-kautomata-pipeline-ui@0.0.0
npm ERR! Found: @angular/core@9.1.12
npm ERR! node_modules/@angular/core
npm ERR!   @angular/core@"^9.1.4" from the root project
npm ERR!
npm ERR! Could not resolve dependency:
npm ERR! peer @angular/core@"7.2.16" from @angular/http@7.2.16
npm ERR! node_modules/@angular/http
npm ERR!   @angular/http@"^7.2.11" from the root project
npm ERR!
npm ERR! Fix the upstream dependency conflict, or retry
npm ERR! this command with --force, or --legacy-peer-deps
npm ERR! to accept an incorrect (and potentially broken) dependency resolution.
```

What problem are we trying to solve?

If I send you a python script, and I used pandas v 3.0.1 and there is a feature that is not in v 2.0.5. And you have v 2.0.5 you will have to upgrade.

However, your code in v 2.0.5 uses something that had a breaking change in v 3.0.1. So you can run my code, but not yours.



dataquest.io

The Solution, Virtual environments.

What are Virtual Environments?

A virtual environment is a self-contained location that isolates Python projects, allowing you to manage dependencies, versions, and packages without conflicts across different projects.

You will learn more about virtual environments in your reading

Install packages in a virtual environment using pip and venv

This guide discusses how to create and activate a virtual environment using the standard library's virtual environment tool `venv` and install packages. The guide covers how to:

- Create and activate a virtual environment
- Prepare pip
- Install packages into a virtual environment using the `pip` command
- Use and create a requirements file

Note

This guide applies to supported versions of Python, currently 3.8 and higher.

Note

When should I use a virtual environment?

Always and forever.

If you do not use a snippet manager, I would suggest you use one. They all work about the same, there is an article for you to read in this week's lessons on them.

I have patterns that I always use, everytime that help me make sure I am producing a consistent result. For Python it is as follows:

- 1) Create my repo
- 2) Instantiate my Virtual Environment
- 3) Create my requirements.txt file and add any packages I know I am going to use
- 4) Run: `pip install -r requirements.txt`
- 5) Create [main.py](#) and paste in some boilerplate code.

Consistency matters

The other pattern that I always use when I am working in a shared repo is:

- 1) Pull main
- 2) Create my local branch from main
- 3) Make my changes in my local branch
- 4) Switch back to main, and re-pull main for any updates that may have been made by teammates while I was working.
- 5) Switch back to my local branch
- 6) Merge main into my local branch
- 7) Resolve any conflicts locally
- 8) Push my branch to github
- 9) PR, once approved delete branch, rinse and repeat.

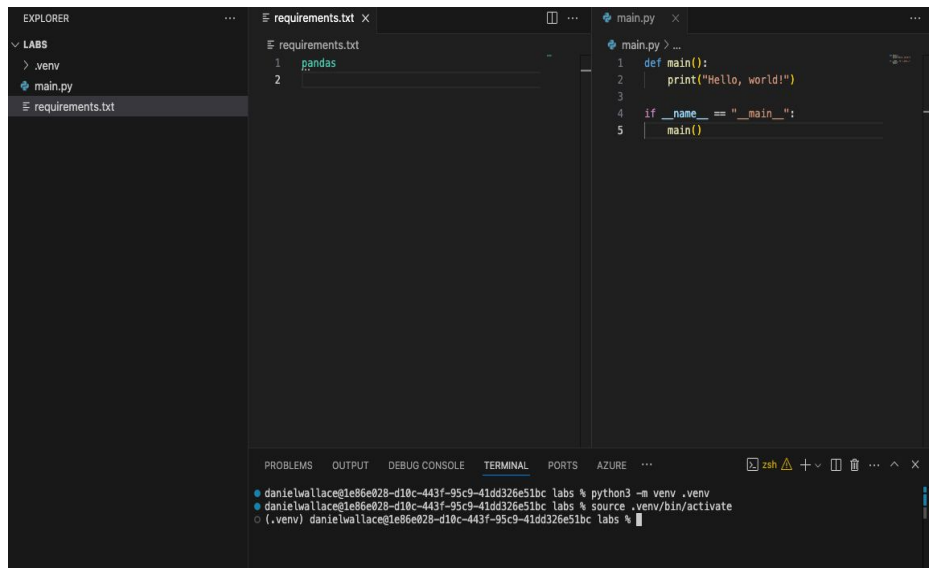
I do this every time, and I do not have any issues in github with conflicts or overwriting anyones changes.

Virtual Environment Expectation

It is the expectation that everything you do, you run in a virtual environment.

When a lab is graded, I will check for the virtual environment file, and run it to make sure it works.

I will install any dependencies from your requirements.txt file, and run your code.



The screenshot shows a Visual Studio Code interface with a dark theme. On the left, the Explorer sidebar shows a file tree with 'LABS' expanded, containing '.venv', 'main.py', and 'requirements.txt'. The main editor area has two tabs: 'requirements.txt' and 'main.py'. The 'requirements.txt' tab is active, showing the following content:

```
requirements.txt
1 pandas
2
```

The 'main.py' tab is also visible, showing the following content:

```
main.py
1 def main():
2     print("Hello, world!")
3
4 if __name__ == "__main__":
5     main()
```

At the bottom, the TERMINAL panel is open, showing a shell prompt and the following commands and output:

```
python3 -m venv .venv
source .venv/bin/activate
(.venv) danielwallace@1e86e028-d10c-443f-95c9-41dd326e51bc labs %
```

That is all